International Journal of Electronics
Vol. **, No. **, ** 2006, 1–18

On the Design of LPM Address Generators Using Multiple LUT Cascades on FPGAs

Hui Qin[†a)], Tsutomu Sasao[†b)], and Jon T. Butler[‡c)]
[†] Department of Computer Science and Electronics, Kyushu Institute of Technology, 680–4, Kawazu, Iizuka, Fukuoka, 820–8502, Japan
[‡] Department of Electrical and Computer Engineering, Naval Postgraduate School, Code EC/Bu, Monterey, CA 93943-5121
(Received November 26, 2006)

We propose the *multiple LUT cascade* as a means to configure an n-input *LPM (Longest Prefix Match) address generator* commonly used in routers to determine the output port given an address. The LPM address generator accepts n-bit addresses which it matches against k stored prefixes. We implement our design on a Xilinx Spartan-3 FPGA for $n = 32$ and $k = 504 \sim 511$. Also, we compare our design to a Xilinx proprietary TCAM (ternary content-addressable memory) design and to another design we propose as a likely solution to this problem. Our best multiple LUT cascade implementation has 5.17 times more throughput, 40.71 times more throughput/area and is 2.97 times more efficient in terms of *area-delay* product than Xilinx's proprietary design, but its area is only 15% of Xilinx's design. Furthermore, we derive a method to determine the optimum configuration of the multiple LUT cascade on an FPGA.

Keywords: LPM address generator; Multiple LUT cascade; FPGA

1 Introduction

The need for higher internet speeds is likely to be the subject of intense interest for many years to come. A network's speed is directly related to the speed with which a node can switch a packet from an input port to an output port. This, in turn, depends on how fast a packet's address can be accessed in memory. The **longest prefix match** (**LPM**) problem is one of determining the output port address from a list of **prefix vectors** stored in memory. For example, if the prefix vector 01001**** is stored in memory, then the packet address 010011111 matches this entry. That is, each bit in the packet address matches

Email: [a)]qinhui@aries01.cse.kyutech.ac.jp; [b)] sasao@cse.kyutech.ac.jp; [c)] jon_butler@msn.com

International Journal of Electronics ISSN 0020-7217 print/ ISSN 1362-3060 online ©2006 Taylor & Francis Ltd
http://www.tandf.co.uk/journals
DOI:10.1080/00207210xxxxxxxxxxxx

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
NOV 2006		

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
On the Design of LPM Address Generators Using Multiple LUT Cascades on FPGAs	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Department of Electrical and Computer Engineering,Monterey,CA,93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**18**	

exactly the corresponding bit in the prefix vector or there is a * or *don't care* in that position. If other stored prefixes match the packet address, then the prefix with the least *don't care* values determines the output port address. That is, the memory entry corresponding to the longest prefix match determines the output port.

An ideal device for this application is a **ternary content-addressable memory** (TCAM) (Pagiamtzis *et al.* 2006, Song *et al.* 2005). The descriptor "ternary" refers to the three values stored, 0, 1, and *. In (Kasnavi *et al.* 2005), the authors proposed pipelined TCAMs for the longest prefix match to increase TCAM efficiency. In (Wang *et al.* 2005), the authors used a TCAM and a small DRAM for the longest prefix match to reduce the required size of TCAM. Unfortunately, TCAM still dissipates more power than standard RAM (Renesas 2005).

Several authors have proposed the use of standard RAM in LPM design. Gupta, Lin, and McKeown showed a mechanism to perform LPM every memory access (Gupta *et al.* 1998). Dharmapurikar, Krishnamurthy, and Taylor have proposed the use of Bloom filters to solve the LPM problem (Dharmapurikar *et al.* 2003). Sasao and Butler have shown that a fast, power-efficient TCAM realization using a look-up table (LUT) cascade (Sasao *et al.* 2006).

In this paper, we propose an extension to the LUT cascade realization: a *multiple LUT cascade* realization that consists of multiple LUT cascades connected to a special encoder. This offers even more efficient realizations in an architecture that is more easily reconfigured when additional prefix vectors are placed in the prefix table.

We have implemented six LPM address generators on the Xilinx Spartan-3 FPGA (XC3S4000-5): Four using multiple LUT cascades, one using Xilinx's TCAM realization based on the Xilinx IP core, and one using registers and gates. In addition, we compare the six LPM address generators on the basis of delay, *delay-area* product, throughput, throughput/area, and FPGA resources used.

A preliminary version of this paper was presented at ARC2006 (Qin *et al.* 2006). We extend these results by introducing the optimum configuration of the multiple LUT cascade, and by showing how to realize the optimum multiple LUT cascade on an FPGA.

The rest of the paper is organized as follows: Section 2 describes the multiple LUT cascade. Section 3 shows other realizations for the LPM address generators. Section 4 presents the implementations of the LPM address generator using an FPGA. Section 5 shows the experimental results. Section 6 discusses the optimum configuration of the multiple LUT cascade implemented on an FPGA. And finally, Section 7 concludes the paper.

Table 1. LPM table.

Address	Prefix Vector
1	1000
2	010*
3	01**
4	1***
5	0***

Table 2. LPM function.

Input	Output Address	Input	Output Address
0000	5	1000	1
0001	5	1001	4
0010	5	1010	4
0011	5	1011	4
0100	2	1100	4
0101	2	1101	4
0110	3	1110	4
0111	3	1111	4

2 Multiple LUT cascades

2.1 *LPM address generators*

A content-addressable memory (CAM) (Shafai *et al.* 1998) stores 0's and 1's and produces the address of the given data. A TCAM, unlike a CAM, stores 0's, 1's, and *'s, where * is a *don't care* value that matches both 0 and 1.

TCAMs are extensively used in routing tables for the internet. A routing table specifies an interface identifier corresponding to the longest prefix that matches an incoming packet, in a process called **Longest Prefix Match (LPM)**. In the LPM table, the ternary vectors have restricted patterns: the prefix consists of only 0's and 1's, and the postfix consists of only *'s (*don't cares*). In this paper, this type of vector is called a **prefix vector**.

Definition 2.1 An n-input m-output k-entry **LPM table** stores k n-element prefix vectors. To assure that the longest prefix address is produced, TCAM entries are stored in descending prefix length, and the first match starting from the top of the table determines the LPM table's output. An address is an m-element binary vector for $m = \lceil \log_2(k+1) \rceil$, where $\lceil a \rceil$ denotes the smallest integer greater than or equal to a. The corresponding **LPM function** is a logic function $f : B^n \rightarrow B^m$, where $f(\vec{x})$ is the smallest address of an entry that is identical to $\vec{x}$ except possibly for don't care values. If no such entry exists, $f(\vec{x}) = 0^m$. The **LPM address generator** is a circuit that realizes the LPM function.

Example 2.2 Table 1 shows an LPM table with 5 4-element prefix vectors. Table 2 shows the corresponding LPM function. It has 16 entries, one for each 4-bit input. The output address is stored for each input corresponding to the address of the longest prefix vector that matches it. □

2.2 *An LUT cascade realization of LPM address generators*

An LPM function, such as that shown in Table 2, can be realized by a single memory which operates as a programmable combinational logic circuit.

However, this often requires prohibitively large memory size.

THEOREM 2.3 *(Sasao 2006) An n-input LPM address generator with k prefix vectors can be realized by an LUT cascade, where each cell realizes a p-input, r-output combinational logic function. Let s be the necessary number of levels or cells. Then,*

$$s \leq \left\lceil \frac{n-r}{p-r} \right\rceil, \tag{1}$$

where $p > r$ *and* $r = \lceil \log_2(k+1) \rceil$.

2.3 *LPM address generators using the multiple LUT cascade*

A single LUT cascade realization of an LPM function often requires many levels. Since the delay is proportional to the number of levels in a cascade, we wish to reduce the number of levels. According to (1), if we increase p, the number of inputs to each cell, then the number of levels s is reduced. For each increase by 1 of p, the memory needed to realize the cell is doubled. However, as shown in Figure 1, we can use a multiple LUT cascade to reduce the number of levels s while keeping p fixed. For an n-input LPM function with k prefix vectors, let the number of rails of each LUT cascade be r. First, starting at the top of the LPM table, partition the set of prefix vectors into g groups of $2^r - 1$ vectors each, except the last group, which has $2^r - 1$ or fewer vectors, where $g = \lceil \frac{k}{2^r-1} \rceil$. For each group of prefix vectors, form an independent LPM function. Next, partition the set of n inputs into s groups. The inputs within a group will apply to a single cell within each cascade. Then, realize each LPM function by an LUT cascade. Thus, we need a total of g LUT cascades, where each LUT cascade consists of s cells. Finally, use a **special encoder** to produce the LPM address. Let v_i $(i = 1, 2, ..., g)$ be the i-th input of the special encoder from the i-th LUT cascade, and let v_{out} be the output value of the special encoder. That is, v_i is the output value of the i-th LUT cascade, where its binary output values are viewed as a standard binary number. Similarly, v_{out} is the output of the special encoder, where its binary output values are viewed as a standard binary number. Then, we have the relation:

$$v_{out} = \begin{cases} v_i + (i-1)(2^r - 1) & \text{if } v_i \neq 0 \text{ and } v_j = 0 \text{ for all } 1 \leq j \leq i-1 \\ 0 & \text{if } v_i = 0 \text{ for all } 1 \leq i \leq g. \end{cases}$$

Note that v_{out} is the position of a prefix vector v in the complete LPM table, while i is the index to the LUT cascade storing v. $(i-1)(2^r-1)$ is the position in the LPM table of the last entry of the previous $(i-1)$-th LUT cascade or

is 0 in the case of the first LUT cascade. Adding v_i to this yields the position of v in the complete LPM table.

Example 2.4 Consider an n-input LPM function with k prefix vectors. When $k = 1000$ and $n = 32$, by Theorem 2.3, we have $r = 10$. Let $p = r + 1 = 11$. When we use a single LUT cascade to realize the function, by Theorem 2.3, we need $\lceil \frac{n-r}{p-r} \rceil = 22$ cells, and the number of levels of the LUT cascade is also 22. Since each cell has 11 address lines and 10 outputs, the total memory size needed to realize the cascade is $2^{11} \times 10 \times 22 = 450,560$ bits. Note that the memory size of each cell, $2^{11} \times 10 = 20,480$ bits, is too large to be realized by a single block_RAM (BRAM) of our FPGA, which stores $18,432$ bits.

However, if we use a multiple LUT cascade to realize the function, we can reduce the number of levels and the total memory. Also, the cells will fit into the BRAMs in the FPGAs. Partition the set of vectors into two groups, and realize each group independently; this requires two LUT cascades. For each LUT cascade, the number of vectors is 500, so we have $r = 9$. Also, let $p = r + 2 = 11$. Then, we need $\lceil \frac{n-r}{p-r} \rceil = 12$ cells in each cascade. Note that the number of levels of the LUT cascades is 12, which is smaller than the 22 needed in the single LUT cascade realization. Since each cell consists of a memory with 9 outputs and at most 11 address lines, the total memory size is at most $2^{11} \times 9 \times 12 \times 2 = 442,368$ bits. Also, note that the size of the memory for a single cell is $2^{11} \times 9 = 18,432$ bits. This fits exactly in the BRAMs of the FPGAs.

Thus, the multiple LUT cascade not only reduces the number of levels and the total memory, but also reduces the size of cells to fit into the available memory in the FPGAs. □

Fig. 1 shows the **multiple LUT cascade realization**. It consists of multiple LUT cascades and a special encoder. The inputs of each LUT cascade are common with other LUT cascades, while the outputs of each LUT cascade are connected to the special encoder. Each LUT cascade realizes an LPM function, while the special encoder generates the LPM address from the outputs of cascades.

The detailed design of each LUT cascade is shown in Fig. 2. Here $\vec{x}_i$ $(i = 1, 2, ..., s)$ denotes the primary inputs to the i-th cell, $\vec{d}_i$ $(i = 1, 2, ..., s)$ denotes the data inputs to the i-th cell and provides the data value to be written in the RAM of the i-th cell, r denotes the number of rails, where $r \leq \lceil \log_2(\frac{k}{g} + 1) \rceil$, $\vec{c}_j$ $(j = 2, 3, ..., s)$ denotes the additional inputs to the j-th cell and is used to select the RAM location along with $\vec{x}_j$ for write access. Note that $\vec{c}_j$ and $\vec{d}_i$ are represented by r bits. All RAMs except perhaps the last one have p address lines; the last RAM has at most p address lines. When *WE* is high, $\vec{c}_j$ is connected to the RAM through a MUX, allowing data to be written into

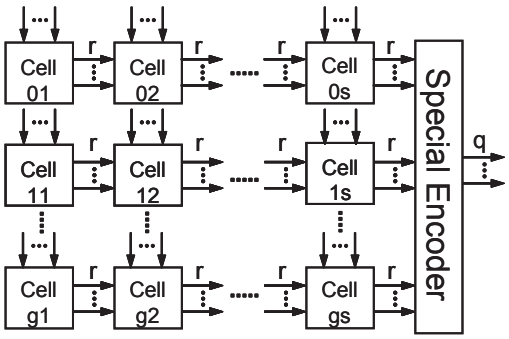

Figure 1. Architecture of the multiple LUT cascade.

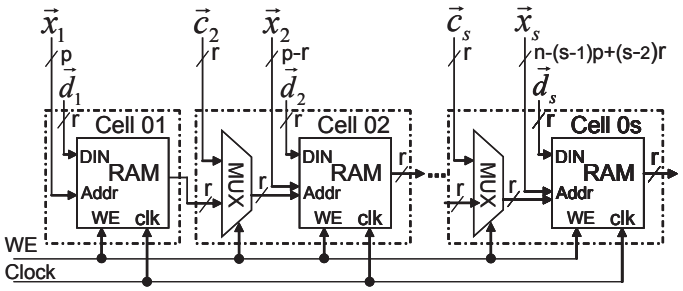

Figure 2. Detailed design of the LUT cascade.

Table 3. 6-entry LPM table.

Address	Prefix Vector
1	100000
2	10010*
3	1010**
4	101***
5	10****
6	1*****

Table 4. Truth table for the corresponding LPM function.

Input						Output			LUT
x_1	x_2	x_3	x_4	x_5	x_6	out_2	out_1	out_0	Cascade
1	0	0	0	0	0	0	0	1	Upper
1	0	0	1	0	*	0	1	0	Cells 1
1	0	1	0	*	*	0	1	1	and 2
1	0	1	1	*	*	1	0	0	Lower
1	0	0	*	*	*	1	0	1	Cells 3
1	1	*	*	*	*	1	1	0	and 4

the RAMs. When *WE* is low, the outputs of the RAMs are connected to the inputs of the succeeding RAMs through a MUX, and the circuit is a cascade that realizes the LPM function. Note that the RAMs are synchronous RAMs. Therefore, the LUT cascade resembles a shift register.

Example 2.5 Table 3 shows a 6-input 3-output 6-entry LPM table, and the truth table of the corresponding LPM function is shown in Table 4. Note that the entries in the two tables are similar. Table 4 is a *compact* truth table, showing only non-zero outputs. Its input combinations must be disjoint. Thus, the two tables are the same except for three entries.

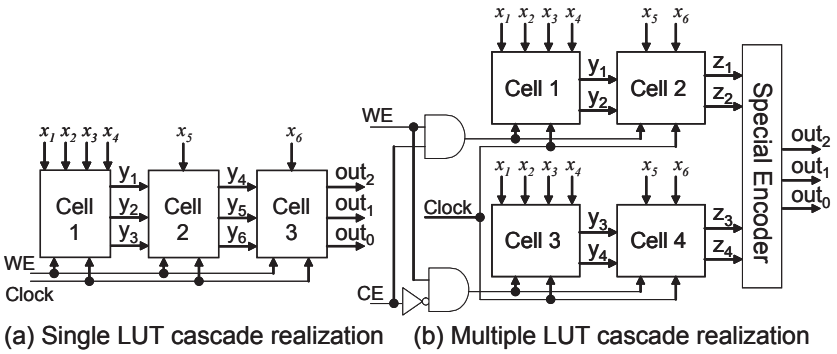

Figure 3. Single LUT cascade realization and the multiple LUT cascade realization.

Single Memory Realization: The number of address lines is 6, and the number of outputs is 3. Thus, the total amount of memory is $2^6 \times 3 = 192$ bits.

Single LUT Cascade Realization: Since there are $k = 6$ prefix vectors, by Theorem 2.3, the number of rails is $r = \lceil \log_2 (6 + 1) \rceil = 3$. Let the number of address lines for the memory in a cell be $p = 4$. By partitioning the inputs into three disjoint sets $\{x_1, x_2, x_3, x_4\}$, $\{x_5\}$, and $\{x_6\}$, we have the cascade in Fig. 3 (a). For simplicity, only the signal lines for the cascade realization are shown. Other lines, such as for storing data, are omitted.

The total amount of memory is $2^4 \times 3 \times 3 = 144$ bits, and the number of levels is $s = 3$. Note that the single LUT cascade requires 75% of the memory needed in the single memory realization.

Multiple LUT Cascade Realization: Partition Table 3 into two parts, each with three prefix vectors. The number of rails in the LUT cascades associated with each separate LPM table is $\lceil \log_2 (3 + 1) \rceil = 2$. Let the number of address lines for the memory in a cell be $p = 4$. By partitioning the inputs into two disjoint sets $\{x_1, x_2, x_3, x_4\}$ and $\{x_5, x_6\}$, we obtain the realization in Fig. 3 (b). The upper LUT cascade realizes the upper part of the Table 4, while the lower LUT cascade realizes the lower part of the Table 4. The contents of each cell is shown in Table 5.

Let v_1 be the output value of the upper LUT cascade, let v_2 be the output value of the lower LUT cascade, and let v_{out} be the output value of the special encoder. Then, in Table 5, (z_1, z_2) viewed as a standard binary number, has value v_1, while (z_3, z_4) viewed as a standard binary number, has value v_2. The special encoder generates the LPM address from the pair of outputs, (z_1, z_2) and (z_3, z_4) :

$$out_2 = \bar{z}_1 \bar{z}_2 (z_3 \vee z_4),$$

Table 5. Truth tables for the cells in the multiple LUT cascade realization.

Cell 1 and Cell 2 (upper LUT cascade)												Cell 3 and Cell 4 (lower LUT cascade)											
x_1	x_2	x_3	x_4	y_1	y_2	x_5	x_6	z_1	z_2	v_1	v_{out}	x_1	x_2	x_3	x_4	y_3	y_4	x_5	x_6	z_3	z_4	v_2	v_{out}
1	0	0	0	0	0	0	0	0	1	1	001	1	0	1	1	0	0	*	*	0	1	1	100
1	0	0	1	0	1	0	*	1	0	2	010	1	0	0	*	0	1	*	*	1	0	2	101
1	0	1	0	1	0	*	*	1	1	3	011	1	1	*	*	1	0	*	*	1	1	3	110
Other values				1	1	*	*	0	0	0	†	Other values				1	1	*	*	0	0	0	†
				Other values				0	0	0	†					Other values				0	0	0	†

† depends on values from the other LUT cascade.

$$out_1 = z_1 \vee \bar{z}_2 z_3 z_4,$$
$$out_0 = z_2 \vee \bar{z}_1 z_3 \bar{z}_4.$$

Note that (out_2, out_1, out_0) viewed as a standard binary number, has value v_{out} corresponding to the address in Table 3. The total memory size is $2^4 \times 2 \times 4 = 128$ bits, and the number of levels is 2. Note that the multiple LUT cascade realization requires 89% of the memory and one fewer level than the single LUT cascade realization. □

3 Other realizations

3.1 *Xilinx's TCAM*

Xilinx (Website of Xilinx) provides a proprietary realization of a TCAM that is produced by the Xilinx CORE Generator tool. Since a TCAM can directly realize an LPM address generator, we compare our proposed multiple LUT cascade realization with Xilinx's TCAM. In the Xilinx CORE Generator 7.1i, we used the following parameters to produce TCAMs.

- *Implementation*: SRL16.
- *Mode*: Standard ternary mode to generate a standard ternary CAM.
- *Depth*: k, the number of words or vectors stored in the TCAM.
- *Data width*: n, the number of bits in words or vectors.
- *Match Address Type*: Binary encoded.
- *Address Resolution*: Lowest.

3.2 *Registers and gates*

We also compare our proposed multiple LUT cascade realization with a direct realization using registers and gates, as shown in Fig 4. We use a register pair (Reg. 1 and Reg. 0) to store each digit of a ternary vector. For example, if the digit is * (*don't care*), the register pair stores (1,1). Thus, for n bit data, we need a $2n$-bit register. The comparison circuit consists of an n-input AND

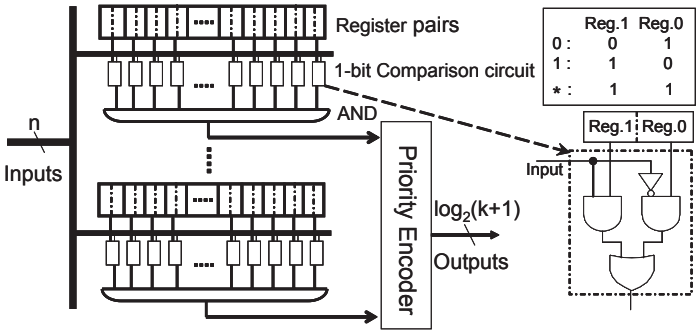

Figure 4. Realization of the address generator with registers and gates.

gate and n 1-bit comparison circuits, each of which produces a 1 if and only if the input bit matches the stored bit or the stored bit is *don't care* (* or 11).

For each prefix vector of an n-input LPM address generator, we need a $2n$-bit register, n 1-bit comparison circuits, and an n-input AND gate. For an n-input address generator with k registered prefix vectors, we need k $2n$-bit registers, nk 1-bit comparison circuits, and k n-input AND gates. In addition, we need a priority encoder with k inputs and $\lceil \log_2 (k+1) \rceil$ outputs to generate the LPM address. If the n-input AND gate is realized as a cascade of 2-input AND gates, this circuit can be considered as a special case of the multiple LUT cascade architecture, where $r = 1$, $p = 2$, and $g = k$. Note that the output encoder circuit is a standard priority encoder.

4 FPGA implementations

We implemented the LPM address generators for 32 inputs and 504∼511 registered prefix vectors on Xilinx Spartan-3 FPGAs (XC3S4000-5) in three ways, the multiple LUT cascade, Xilinx CORE Generator 7.1i, and registers and gates. XC3S4000-5 (Spartan-3 FPGA data sheet 2005) has 96 BRAMs and 27,648 slices. Each BRAM contains 18K bits, and each slice consists of two 4-input LUTs, two D-type flip-flops, and multiplexers. For each implementation, we described the circuit by Verilog HDL, and then used Xilinx ISE 7.1i to synthesize and to perform place and route.

First, we used the multiple LUT cascade to realize the LPM address generators. To use the BRAMs in the FPGA efficiently, we chose the memory size of a cell in the LUT cascade not to exceed the size of a BRAM unit. Let p be number of address lines of the memory in the cell. Since each BRAM contains $2^{11} \times 9$ bits, we have the relation: $2^p \cdot r \leq 2^{11} \times 9$, where r is the number of rails. Thus, we have $p = \lfloor \log_2 (9/r) \rfloor + 11$, where $\lfloor a \rfloor$ denotes the largest integer less than or equal to a.

Table 6. Four multiple LUT cascade realizations.

Design	**Number of prefix vectors**	***r***	***p***	**Group**	**Level**
$r6p11$	504	6	11	8	6
$r7p11$	508	7	11	4	7
$r8p11$	510	8	11	2	8
$r9p11$	511	9	11	1	12

r: Number of rails
p: Number of address lines of the RAM in a cell
Group: Number of LUT cascades

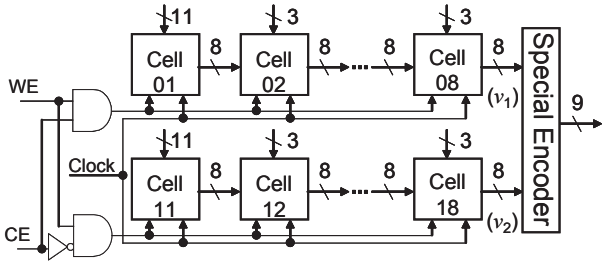

Figure 5. Realization of $r8p11$.

We designed four LPM address generators $r6p11$, $r7p11$, $r8p11$, and $r9p11$, as shown in Table 6, where the column **Number of prefix vectors** denotes the number of registered prefix vectors, the column ***r*** denotes the number of rails, the column ***p*** denotes the number of address lines of the RAM in a cell, the column **Group** denotes the number of LUT cascades, and the column **Level** denotes the number of levels or cells in the LUT cascade.

To explain Table 6, consider $r8p11$ which is shown in Fig 5. For $r8p11$, since the number of rails is $r = 8$, the number of groups is $\lceil \frac{510}{2^8-1} \rceil = 2$. Thus, we need two LUT cascades. Since each LUT cascade consists of 8 cells, the number of levels of $r8p11$ is 8. To efficiently use BRAMs in the FPGA, the number of address lines of the RAM in the cell is set to $p = \lfloor \log_2 (9/8) \rfloor + 11 = 11$. Let v_1 be the value of the outputs of the upper LUT cascade, let v_2 be the value of the outputs of the lower LUT cascade, and let v_{out} be the value of the outputs of the special encoder. Then, we have the relation:

$$v_{out} = \begin{cases} v_2 + 255 & \text{if } \ v_1 = 0 \text{ and } v_2 \neq 0, \\ v_1 & \text{otherwise.} \end{cases}$$

The circuit realizing this expression requires 11 slices on the FPGA. For the whole circuit, $r8p11$ requires 16 BRAMs and 69 slices. From this table, we can see that decreasing r, increases the number of groups, but decreases the number of levels.

Next, we used the Xilinx CORE Generator 7.1i tool to produce Xilinx's TCAM. Since the Xilinx CORE Generator 7.1i does not support TCAMs with

32 inputs and 505∼511 registered prefix vectors, we designed a TCAM with 32 inputs and 504 registered prefix vectors. The resulting TCAM required 8,590 slices. Note that Xilinx's TCAM requires one clock cycle to find a match.

Finally, we designed the LPM address generator with $n = 32$ inputs and $k = 511$ registered prefix vectors using registers and gates, as shown in Fig 4. This design is denoted **Reg-Gates**. Note that the number of inputs is 32 and the number of outputs is 9. This design required 27,646 slices.

5 Performance and comparisons

In Table 7, we show the performance of multiple LUT cascade realizations (i.e., $r6p11$, $r7p11$, $r8p11$, and $r9p11$), and compare them with Xilinx's TCAM and Reg-Gates. In Table 7, the column **Level** denotes the number of levels or cells in the LUT cascade, the column **Slice** denotes the number of occupied slices, the column **Memory** denotes the amount of memory required, and the column **F_clk** denotes the maximum clock frequency. The column **tco** denotes the maximum clock-to-output propagation delay. (It is the maximum time required to obtain a valid output at the output pin that is fed by a register after a clock signal transition on an input pin that clocks the register). The column **tpd** denotes the maximum propagation time from the inputs to the outputs. The column **Th.** denotes the maximum throughput. Since the LPM address generator has 9 outputs, it is calculated as:

$$\text{Th.} = 9 \cdot \text{F\_clk}.$$

For Reg-Gates, **Delay** denotes the maximum delay from the input to the output and is equal to **tpd**. For multiple LUT cascade realizations and Xilinx's TCAM, **Delay** denotes the total delay, and is calculated by:

$$\text{Delay} = \frac{1000 \cdot \text{Level}}{\text{F\_clk}} + tco,$$

where 1000 is a unit conversion factor.

Consider the area occupied by the various realizations. From the Spartan-3 family architecture (Spartan-3 FPGA data sheet 2005), we can see that the area of one BRAM is at least the area of 16 slices (a slice consists of two "4-input LUTs", two flip-flops, and miscellaneous multiplexers).

An alternative estimate shows that the area of one BRAM is equivalent to that of 96 slices, as follows. In the Xilinx Virtex-II FPGA, one "4-input LUT" occupies approximately the same area as 96 bits of BRAM (also containing 18K bits) (Sproull *et al.* 2005). Note that both "4-input LUTs" and BRAMs of the Virtex-II FPGA are similar to those of the Spartan-3 FPGA. Thus,

Table 7. Comparisons of FPGA implementations of the LPM address generator.

Design	**Level**	**Slice**	**Memory** (BRAM)	**F_clk** (MHz)	**tco/tpd** (ns)	**Th.** (Mbps)	**Area**[a] (slice)	**Th./Area** ($\frac{\text{Mbps}}{\text{slice}}$)	**Delay** (ns)	**Area-Delay** (slice-μs)
$r6p11$	6	178	48	103.89	24.89 (tco)	935	4786	0.195	82.64	395.53
$r7p11$	7	116	28	113.77	23.46 (tco)	1024	2804	0.365	84.99	238.31
$r8p11$	8	69	16	139.93	20.91 (tco)	**1259** (*best*)	1605	0.785	79.57	127.71
$r9p11$	12	99	12	139.08	13.72 (tco)	1252	**1251** (*best*)	**1.001** (*best*)	100.00	**125.10** (*best*)
Xilinx's TCAM	1	8590		22.52	13.48 (tco)	203	8590	0.024	**57.88** (*best*)	497.23
Reg-Gates		27646			58.67 (tpd)		27646		58.67	1621.99

[a]We assume that the area for one BRAM is equivalent to the area of 96 slices.

we can deduce that one BRAM of the Spartan-3 FPGA occupies about the same area as 192 ($= 18 \times 1024/96$) "4-input LUTs". If we view one "4-input LUT" as approximately one-half a slice according to our discussion in the previous paragraph, we conclude that one BRAM has about the same area as 96 ($= 192/2$) slices. Thus, two estimates of the area for one BRAM, 16 and 96 slices are quite different. For this analysis, a worst case of 96 slices/BRAM was used.

In Table 7, the column **Area** denotes the equivalent utilized area, where the area for one BRAM is equivalent to the area for 96 slices. The column **Th./Area** denotes the efficiency of throughput per area for one slice. The column **Area-Delay** denotes the *area-delay* product. The value denoted by *best* shows the best result.

Xilinx's TCAM has the smallest delay, but requires many slices. Reg-Gates has almost the same delay as Xilinx's TCAM, but requires about three times as many slices as Xilinx's TCAM. Note that Reg-Gates requires no clock pulses in the LPM address generation operation, while the others are sequential circuits that require clock pulses. Since the delay of Reg-Gates is 58.67 ns, the equivalent throughput is $(1000/58.67) \times 9 = 153$ (Mbps), which is lower than all others.

All multiple LUT cascade realizations have higher throughput, smaller area, higher throughput/area, and are more efficient in terms of *area-delay* than Xilinx's TCAM. $r8p11$ has the highest throughput and the smallest delay among all multiple LUT cascade realizations, but is slightly less efficient in terms of *area-delay* than $r9p11$. $r9p11$ has the smallest area, the highest throughput/area, and the highest efficiency in terms of *area-delay* among all realizations. Although $r8p11$ has almost the same *area-delay* as $r9p11$, its area is 28% more larger than that of $r9p11$. Hence, $r9p11$ is the best multiple LUT cascade realization since it has 5.17 times more throughput, 40.71 times more throughput/area, and is 2.97 times more efficient in terms of *area-delay* product than Xilinx's TCAM, while the area is only 15% of Xilinx's TCAM.

Table 8. *Memory-Delay* for multiple LUT cascade realizations.

Design	$r6p11$	$r7p11$	$r8p11$	$r9p11$
Area-Delay (slice-μs)	395.53	238.31	127.71	**125.10**
Memory-Delay (BRAM-μs)	3.97	2.38	1.27	**1.20**

6 The optimum configuration of the multiple LUT cascade

Firstly, consider the relation between the required memory size and the total area. As can be seen from Table 7, for $r6p11$, which has the most complicated encoder, the memory required occupies 96.3% ($=\frac{48\times96}{4786}$) of the total area. For $r9p11$, the memory required occupies 92.1% ($=\frac{12\times96}{1251}$) of the total area. Note that $r9p11$ has the smallest proportion of the area for memory to the total area among all the multiple LUT cascade realizations. Thus, the memory consumes no less than 92% of the total area. In addition, as shown in Table 7, the size of *Memory* is approximately proportional to the *Area*. Hence, we can assume that the multiple LUT cascade realization with the smallest memory size corresponds to that with the smallest total area. Secondly, consider the relation between *Area-Delay* and *Memory-Delay* product. As shown in Table 8, $r9p11$ has both the smallest *Area-Delay* and the smallest *Memory-Delay* among all multiple LUT cascade realizations. Note that the value of *Memory-Delay* is approximately proportional to the *Area-Delay*. Thus, we can assume that the realization with the smallest *Memory-Delay* corresponds to that with the smallest *Area-Delay*. Therefore, we can use the total size of memory required instead of the total area, and *Memory-Delay* instead of *Area-Delay* to find the optimum multiple LUT cascade realization. Doing this allows a formal analysis, as shown in the next section.

6.1 *Total size of memory*

Consider the multiple LUT cascade implementation of an n-input LPM address generator that stores k prefix vectors. Let each cell be realized as a reconfigurable memory with m bits. For the implementations discussed previously in this paper, this memory is a BRAM of the Spartan-3 FPGA, where $m = 18,432$ bits. Each cell in the LUT cascade has r outputs, where $r \leq \lceil \log_2(k+1) \rceil$. With m bits stored in each memory and r bits per word, $\frac{m}{r}$ words are stored in each LUT cell. Therefore, the number of address inputs for each LUT cell is $p(r) = \lfloor \log_2 \frac{m}{r} \rfloor$. Note that $r \leq p(r) - 1$. Let $M(r)$ be the total memory needed to implement the given LPM address generator. That is,

$$M(r) = msg, \tag{2}$$

where $s = \lceil \frac{n-r}{p(r)-r} \rceil$ is the number of cells in each of the $g = \lceil \frac{k}{2^r-1} \rceil$ cascades that make up the multiple LUT cascade realization of the LPM address generator.

THEOREM 6.1 *$M(r)$ is a monotone decreasing function of r for $r \leq p(r) - 2$.*

Since $M(r)$ is monotone decreasing for $r \leq p(r) - 2$, to find the minimum $M(r)$, it is only necessary to find $M(r)$ for $r = p(r) - 2$ and $r = p(r) - 1$, an upper bound in r.

6.2 *Memory-delay product*

From Table 7, we observed that the delay in an n-input LPM address generator is given approximately as

$$D = \frac{1000}{F\_clk}(s + 2), \tag{3}$$

where F_clk is the frequency of the clock in MHz. Let $MD(r)$ be the memory-delay product of the multiple LUT cascade realization of the address generator. Therefore,

$$MD(r) = (msg)(\frac{1000}{F\_clk}(s + 2)), \tag{4}$$

THEOREM 6.2 *$MD(r)$ is a monotone decreasing function of r for $r \leq p(r)-5$. Specially, when $p(r-1) = p(r)$, $MD(r)$ is a monotone decreasing function of r for $r \leq p(r) - 3$.*

Since $MD(r)$ is monotone decreasing for $r \leq p(r) - 5$, to find the minimum $MD(r)$, it is only necessary to find $MD(r)$ for $r = p(r) - i$, where i=1, 2, ..., 5, or for five values of r. Specially, when $p(r-1) = p(r)$, to find the minimum $MD(r)$, we only need to consider three cases for $r = p(r) - i$, where i=1, 2, and 3.

6.3 *Optimum multiple LUT cascade for the BRAM containing 18K bits, 16K bits or 4K bits*

In popular FPGAs, such as Xilinx's FPGAs or Altera's FPGAs, the sizes of BRAM are 18K bits or 4K bits. For other FPGAs, the size of BRAM can be 16K bits. We consider these three types of BRAMs in the following discussion.

For 16K-bit BRAM, from $p(r) = \lfloor \log_2 (16384/r) \rfloor$, we have $p(r) = 10$ for $r = 9$ and $p(r) = 11$ for $5 \leq r \leq 8$. Since $p(r-1) = p(r) = 11$ for $5 \leq r \leq 8$,

Table 9. The value of r that makes $MD(r)$ minimum.

Block_RAM size	The minimum *Memory-Delay*
18K bits	$r = r\_max$ when $r\_max \leq 8$ $r = r\_optimal$ when $r\_max = 9$
4K bits	$r = r\_max$ when $r\_max \leq 6$ $r = r\_optimal$ when $r\_max = 7$
16K bits	$r = r\_max$ for $r \leq 8$

r_max is the maximum integer r that satisfies both $r \leq p(r) - 1$ and $r \leq \lceil \log_2 (k + 1) \rceil$, where $p(r) = \lfloor \log_2 \frac{m}{r} \rfloor$ and m denotes the size of a BRAM.

$r\_optimal$ is r that makes $s \cdot g \cdot (s+2)$ minimum, where $s = \lceil \frac{n-r}{p(r)-r} \rceil$ and $g = \lceil \frac{k}{2^r-1} \rceil$. For m = 18K-bit BRAM, $r\_optimal$ can be obtained by calculating values only for $r = p(r) - 2 = 9$ and $r = p(r)-3 = 8$. For m = 4K-bit BRAM, $r\_optimal$ can be obtained by calculating values only for $r = p(r)-2 = 7$ and $r = p(r)-3 = 6$.

from Theorem 6.2, $MD(r)$ decreases with r when $1 \leq r \leq (11 - 3) = 8$. Let $\zeta(r) = s \cdot g \cdot (s + 2)$, where $s = \lceil \frac{n-r}{p(r)-r} \rceil$ and $g = \lceil \frac{k}{2^r-1} \rceil$. We can verify $\zeta(8) < \zeta(9)$ when $n > 15$. In most applications, we can assume that $n > 16$. Thus, we can conclude that $MD(r)$ is minimum when r is maximum, where $r \leq 8$.

When the size of a BRAM is m = 18K bits, from $p(r) = \lfloor \log_2 (9/r) \rfloor + 11$, we have $p(r) = 11$ for $5 \leq r \leq 9$. When m = 4K bits, from $p(r) = \lfloor \log_2 (8/r) \rfloor + 9$, we have $p(r) = 9$ for $5 \leq r \leq 8$ and $p(r) = 10$ for $r = 4$. Thus, for both m = 18K-bit and m = 4K-bit, we have $p(r - 1) = p(r)$ when $r \leq p(r) - 5$. From Theorem 6.2, $MD(r)$ is minimum when $r = 11-2 = 9$ or $r = 11-3 = 8$ for 18K-bit BRAM, and $r = 9-1 = 8$, $r = 9-2 = 7$, or $r = 9-3 = 6$ for 4K-bit BRAM. We can verify $\zeta(6) < \zeta(8)$ for 4K-bit BRAM when $n > 14$. In most applications, we can assume that $n > 16$. Thus, we only need to consider the case of $r = p(r) - 2$. Depending on the values of n and k, $MD(r)$ is minimum when $r = p(r) - 2$ or $r = p(r) - 3$. However, for an LPM address generator with fixed n and k, we can easily obtain an r that minimizes $\zeta(r) = s \cdot g \cdot (s+2)$ by calculating the values for $r = p(r) - 2$ and $r = p(r) - 3$.

Table 9 shows the values of r that minimize $MD(r)$ for three types of BRAMs.

In Table 7, *Area-Delay* for $r9p11$ and $r8p11$ are nearly the same. Note that when $p = 11$ and $n = 32$, $\zeta(p - 2) = 0.30$ and $\zeta(p - 3) = 0.31$ are almost the same value.

For BRAM containing 18K bits, Theorem 6.1 shows that the area for $r = 9$ is smaller than for $r = 8$. If $MD(r)$ for $r = 9$ is almost the same as for $r = 8$, then the multiple LUT cascade is optimum when $r = 9$. Similar to 4K-bit BRAM, if $MD(r)$ for $r = 7$ and $r = 6$ are almost the same, then the multiple LUT cascade is optimum when $r = 7$. The following example shows the design of an optimum multiple LUT cascade.

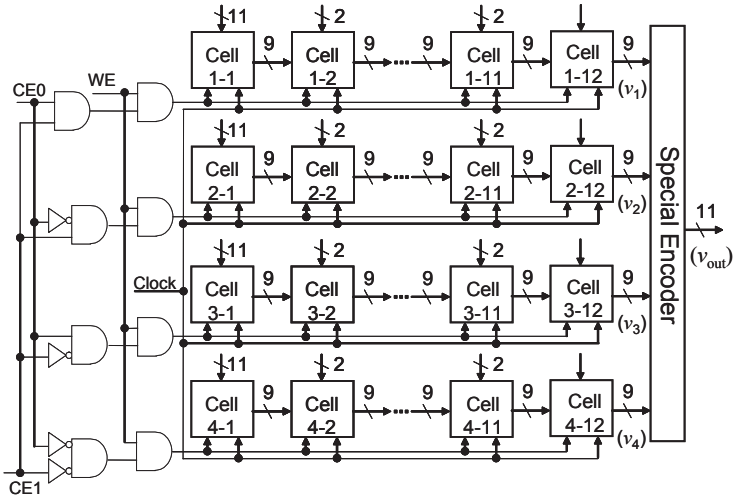

Figure 6. Optimum realization of $r9p11g4$.

Example 6.3 Consider an LPM address generator with $n = 32$ and $k = 2040$ implemented on a Spartan-3 FPGA. Note that the size of a BRAM is $m = 18$K bits. First, from $p(r) = \lfloor \log_2 (9/r) \rfloor + 11$, we have $p(r) = 11$ when $5 \leq r \leq 9$. To obtain the optimal *Area-Delay* realization, from Table 9, r can be 8 or 9 when $n = 32$ and $k = 2040$. Let $\zeta(r) = s \cdot g \cdot (s + 2)$. We have $\zeta(9) = 672$, and $\zeta(8) = 640$. Note that $\zeta(9)$ is nearly the same as $\zeta(8)$. Since the area for $r = 9$ is minimum from Theorem 6.1, the multiple LUT cascade is optimum when $p = 11$ and $r = 9$. In this case, a realization with $g = \lceil \frac{k}{2^r - 1} \rceil = \lceil \frac{2044}{2^9 - 1} \rceil = 4$ LUT cascades is optimum. Also, the number of levels is $\lceil \frac{n-r}{p-r} \rceil = \lceil \frac{32-9}{11-9} \rceil = 12$, which shows that each LUT cascade consists of 12 cells. Finally, we need a special encoder. Let v_1 be the value of the outputs of the top LUT cascade, let v_2 be the value of the outputs of the second LUT cascade, let v_3 be the value of the outputs of the third LUT cascade, let v_4 be the value of the outputs of the fourth LUT cascade, and let v_{out} be the value of the outputs of the special encoder. Then, we have the relation:

$$v_{out} = \begin{cases} v_4 + 1533 & \text{if } v_1 = v_2 = v_3 = 0 \text{ and } v_4 \neq 0, \\ v_3 + 1022 & \text{if } v_1 = v_2 = 0 \text{ and } v_3 \neq 0, \\ v_2 + 511 & \text{if } v_1 = 0 \text{ and } v_2 \neq 0, \\ v_1 & \text{otherwise.} \end{cases}$$

The optimum realization of $r9p11g4$ is shown in Fig 6. □

We also implemented the LPM address generator with n=32 and k=2040 on Xilinx Spartan-3 FPGAs (XC3S4000-5). Table 10 shows that $r9p11g4$ has

Table 10. FPGA implementations of the LPM address generator with n=32 and k= 2040.

Design	**Level**	**Slice**	**Memory** (BRAM)	**F_clk** (MHz)	**tco** (ns)	**Th.** (Mbps)	**Area**[a] (slice)	**Th./Area** ($\frac{\text{Mbps}}{\text{slice}}$)	**Delay** (ns)	**Area-Delay** (slice-μs)
$r8p11g8$	8	299	64	111.20	26.00	1223	6443	0.190	**97.94**	**631.04**
$r9p11g4$	12	241	48	111.33	23.00	**1225**	**4849**	**0.253**	130.79	634.19

[a]We assume that the area for one BRAM is equivalent to the area of 96 slices.
$r8p11g8$ denotes the FPGA implementation with r=8, p=11, and g=8.
$r9p11g4$ denotes the FPGA implementation with r=9, p=11, and g=4.

almost the same throughput and *area-delay* as $r8p11g8$, but its area is only 75% of $r8p11g8$. In addition, $r9p11g4$ has higher throughput/area than that of $r8p11g8$. Thus, $r9p11g4$ is the optimum realization for the LPM address generator with n=32 and k=2040.

7 Conclusions

In this paper, we presented the multiple LUT cascade to realize LPM address generators. In addition, we discussed an approach to obtain the optimum configuration of multiple LUT cascade on FPGAs. Although we illustrated the design method for $n = 32$ and $k = 504 \sim 511$, it can be extended to other values of n and k.

We implemented four LPM address generators (i.e. $r6p11$, $r7p11$, $r8p11$, and $r9p11$) on the Xilinx Spartan-3 FPGA (XC3S4000-5) by using the multiple LUT cascade. For comparison, on the same type of FPGA, we also implemented Xilinx's proprietary TCAM and Reg-Gates, an approach proposed by us as a likely solution to the LPM problem. Xilinx's TCAM has the smallest delay, but requires many slices. Reg-Gates has almost the same delay as Xilinx's TCAM, but requires the largest area, and requires about three times as many slices as Xilinx's TCAM. All multiple LUT cascade realizations have higher throughput, smaller area, higher throughput/area and more efficient in terms of *area-delay* product than Xilinx's TCAM.

ACKNOWLEDGMENTS

This research is partly supported by a Grant-in-Aid for Scientific Research from JSPS, MEXT, a grant from Kitakyushu Area Innovative Cluster Project, and by an NSA contract.

REFERENCES

K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, Vol. 41, No. 3, 2006, pp. 712-727.

H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," *Proc. ACM/SIGDA 13th International Symposium on Field Programmable Gate Arrays*, 2005, pp. 238 - 245.

P. C. Wang, C. T. Chan, R. C. Chen, and H. Y. Chang, "An efficient ternary CAMs entry-reduction algorithm for IP forwarding engine," *IEE Proceedings-Communications Vol. 152*, 2005, pp. 172-176.

S. Kasnavi, V. C. Gaudet, P. Berube, and J. N. Amaral, "A novel hardware-based longest prefix matching scheme for TCAMs," *Proc. IEEE International Symposium on Circuits and Systems*, 2005, pp. 3339 - 3342.

Renesas Technology Inc.: 9 M/18 M-bit Full Ternary CAM, Datasheet, February 2005.

P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," *Proc. IEEE INFOCOM*, 1998, pp. 1241-1247.

S. Dharmapurikar, P. Krishnamurthy, and D. Taylor, "Longest prefix matching using Bloom filters," *Proc. ACM SIGCOMM*, 2003, pp. 201-212.

T. Sasao, and J.T. Butler, "Implementation of multiple-valued CAM functions by LUT cascades," *Proc. IEEE International Symposium on Multiple-Valued Logic*, May 2006, CD-ROM.

H. Qin, T. Sasao, and J.T. Butler, "Implementation of LPM address generators on FPGAs," *Proc. International Workshop on Applied Reconfigurable Computing (ARC2006)*, March 2006, pp 170-181.

F. Shafai, K.J. Schultz, G.F.R. Gibson, A.G. Bluschke, D.E. Somppi, "Fully parallel 30-MHz, 2.5-Mb CAM," *IEEE Journal of Solid-State Circuits*, Vol. 33, No. 11, 1998, pp. 1690-1696.

T. Sasao, "Analysis and synthesis of weighted-sum functions," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25, No. 5, 2006, pp. 789-796.

Website of Xilinx is available at http://www.xilinx.com

Xilinx, Inc., "Spartan-3 FPGA family: Complete data sheet," DS099, Aug. 19, 2005.

T. Sproull, G. Brebner, C. Neely, "Mutable codesign for embedded protocol processing," *Proc. IEEE 15th International Conference on Field Programmable Logic and Applications*, 2005, pp. 51-56.